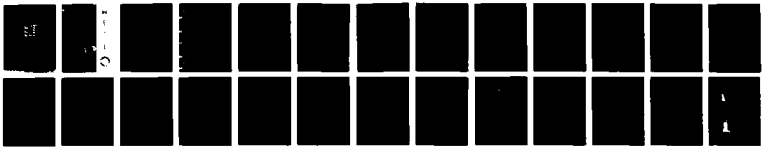
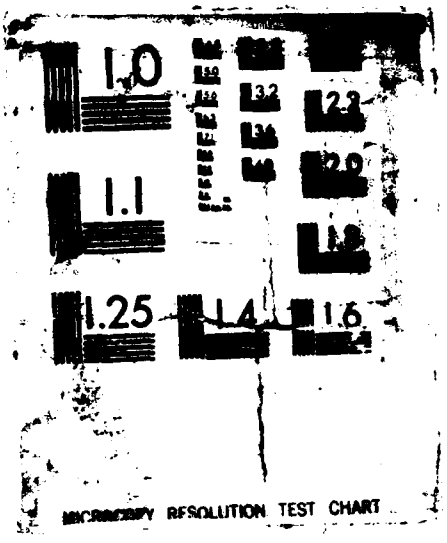
AD-A184 246 A TIGHT UPPER BOUND FOR THE SPEED-UP OF PARALLEL BEST-FIRST BRANCH-AND-BO..(U) MARYLAND UNIV COLLEGE PARK CENTER FOR AUTOMATION RESEARCH S HUANG ET AL. 1/1

UNCLASSIFIED MAY 87 CAR-TR-290 ETL-0462 DACA76-84-C-0004 F/G 12/1 NL



END
9-87
DTIC


1.0
1.1
1.25
1.4
1.6
RESOLUTION TEST CHART

ETL-0462

A tight upper bound for the speed-up of parallel best-first branch-and-bound algorithms

Shie-rei Huang
Larry S. Davis

Center for Automation Research
University of Maryland
College Park, MD 20742-3411

Prepared for
U.S. ARMY CORPS OF ENGINEERS
ENGINEER TOPOGRAPHIC LABORATORIES
FORT BELVOIR, VIRGINIA 22060-5546

Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

A184246

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CAR-TR-290 CS-TR-1852	5. MONITORING ORGANIZATION REPORT NUMBER(S) ETL-0462
6a. NAME OF PERFORMING ORGANIZATION University of Maryland 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION U.S. Army Engineer Topographic Laboratories
6c. ADDRESS (City, State and ZIP Code) Center for Automation Research College Park, MD 20742-3411	7b. ADDRESS (City, State and ZIP Code) Fort Belvoir, VA 22060-5546
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency 8b. OFFICE SYMBOL (If applicable) ISTO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DACA76-84-C-0004
8c. ADDRESS (City, State and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 62301E PROJECT NO.: TASK NO.: WORK UNIT NO.:
11. TITLE (Include Security Classification) A Tight Upper Bound for the Speedup of Parallel Best-First Branch-and-Bound Algorithms	
12. PERSONAL AUTHOR(S) Shie-rei Huang and Larry S. Davis	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO N/A	14. DATE OF REPORT (Yr., Mo., Day) May 1987 15. PAGE COUNT 24
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Most previous studies of the speedup of parallel branch-and-bound algorithms are based on the amount of work done in the parallel case and in the sequential case [14, 17, 18, 23]. Any evaluation of a parallel algorithm should include both the execution time and the synchronization delay [30]. In this paper, a finite population queueing model is used to capture the synchronization delay in parallel branch-and-bound algorithms and to qunatitatively predict the behavior of their speedup. A program to solve the Traveling Salesman Problem was written on a BBN Butterfly [2] multiprocessor to empirically demonstrate the credibility of this theoretical analysis. Finally, we note that similar analyses can be applied to evaluate parallel AI systems in which processes communicate through a shared global database.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Rosalene Holecheck	22b. TELEPHONE NUMBER (Include Area Code) (202) 355-2767 22c. OFFICE SYMBOL ETL-RI-T

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

CAR-TR-290
CS-TR-1852

DACA76-84-C-0004
May 1987

A TIGHT UPPER BOUND FOR THE SPEEDUP OF PARALLEL BEST-FIRST BRANCH-AND-BOUND ALGORITHMS

Shie-rei Huang
Larry S. Davis
Center for Automation Research
University of Maryland
College Park, MD 20742

ABSTRACT

Most previous studies of the speedup of parallel branch-and-bound algorithms are based on the amount of work done in the parallel case and in the sequential case[14,17,18,23]. Any evaluation of a parallel algorithm should include both the execution time and the synchronization delay[30]. In this paper, a finite population queueing model is used to capture the synchronization delay in parallel branch-and-bound algorithms and to quantitatively predict the behavior of their speedup. A program to solve the Traveling Salesman Problem was written on a BBN Butterfly[2] multiprocessor to empirically demonstrate the credibility of this theoretical analysis. Finally, we note that similar analyses can be applied to evaluate parallel AI systems in which processes communicate through a shared global database.

The support of the Defense Advanced Research Projects Agency and the U.S. Army Engineer Topographic Laboratories under Contract DACA76-84-C-0004 is gratefully acknowledged.

1. Introduction

Branch-and-bound is one of the most general methods of solving combinatorial optimization problems. It has wide applications in the fields of operations research[15] and artificial intelligence[21]. In general, the branch-and-bound method repeatedly partitions the solution space into smaller and smaller subspaces, and a lower bound (assuming minimization is to be achieved) of the cost of each subspace is estimated. A subspace will no longer be partioned when its lower bound exceeds the known cost upper bound of the solution space. The first found solution whose cost does not exceed any lower bound of the partioned solution subspaces is an optimal solution. For a review and formal definition of the branch-and-bound method, see references[10,28].

Recently there has been wide interest in parallelizing branch-and-bound or combinatorial search methods on multicomputers. Imai et al.[11] proposed a parallel searching scheme for multiprocessors. Wah et al.[29] described a multicomputer architecture for solving combinatorial search problems. The behavior of parallel branch-and-bound algorithms has been studied by several researchers[14,17,19,23]. Three type of anomalies for the speedup of parallel branch-and-bound algorithms were recognized. The speedup obtained when k processors are used can be (a) greater than k (*acceleration anomalies*), (b) between one and k (*deceleration anomalies*) or (c) less than one (*detrimental anomalies*). However acceleration and detrimental anomalies are unlikely to occur in parallel best-first branch-and-bound algorithms unless there are a large number of subproblems having the same lower bound; in addition, a nearly linear speedup can be achieved for a

large number of processors when the problem size is large[17,18,23].

One of the drawback of previous analyses of parallel branch-and-bound algorithms is that the speedup is based on the amount of work done in the parallel case and in the sequential case. Any evaluation of a parallel algorithm should include both the execution time as well as the the synchronization delay[30]. In this paper, using a finite population queuing model, we quantitatively predict the behavior of the speedup of parallel branch-and-bound algorithms which are parallelized on shared memory multiprocessors. A program to solve the Traveling Salesman Problem was written on a BBN Butterfly^2 multiprocessor to empirically demonstrate the credibility of this analysis.

In Section 2, a finite population queueing model is reviewed. Section 3 describes a general approach, that serves as the basis for this paper, to parallelizing branch-and-bound algorithms on multiprocessors. Section 4 describes how the speedup behavior of parallel branch-and-bound algorithms can be analyzed using the finite population single server queueing model. Section 5 presents simulation results on a BBN Butterfly machine. Finally, a conclusion which includes some suggestions about avoiding early saturation in speedup is provided.

2. Finite Population Queueing Model

We now review a queueing model which has been effectively used to predict the performance of interactive time-sharing computer systems[26]. In Figure 2.1 we have a closed network consisting of a single central server and a finite number of "sources". This queueing model operates in the following way: When a source makes a request at the central server, the source "goes to sleep". The request,

possibly after a queueing delay, then receives service at the central server. When the request is finally served, the response is fed back to the source. At this point, the source "wakes up" and starts generating a new request. The time spent by the source in generating a new request is referred to as its *thinking time.*

We assume that each source has an average thinking time of λ^{-1} sec; or equivalently, each source, when thinking, generates requests at an average rate of λ. The central server has, for each request, an average processing time of μ^{-1} or an average processing rate of μ. We are interested in the response time of the central server as seen by a source.

The average response time $\overline{W}$ can be solved by equating the arrival rate of requests to the central server to the departure rate from the central server, or by Little's Formula[13]:

$$\overline{W} = \frac{n}{\mu(1 - p_0)} - \frac{1}{\lambda} \tag{2.1}$$

where n is the number of sources and p_0 is the probability that there is no outstanding request at the central server. Define $\rho = \frac{\lambda}{\mu}$. Equation (2.1) then can be rewritten as

$$\overline{W} = \frac{1}{\mu}\left(\frac{n}{1 - p_0} - \frac{1}{\rho}\right) \tag{2.2}$$

Since $0 < 1 - p_0 < 1$,

$$\overline{W} > \frac{1}{\mu}\left(n - \frac{1}{\rho}\right) \tag{2.3}$$

When the number of sources is large, p_0 can become very small. Therefore Equation (2.3) can be considered as an asymptotic approximation for $\overline{W}$.

Note that the distributions of the thinking time of the sources and the processing time at the central server do not play a role in the derivation of Equation (2.2). However, the above informal mean flow analysis is strictly true only for certain types of service and thinking time distributions, namely an exponential distribution or rational distribution with round-robin scheduling[13]. Nevertheless, this model has been found to be surprisingly robust in evaluating systems which violate most of the strict assumptions[26].

If we assume that the thinking time and the processing time are both exponentially distributed, then the probability p_0 can be determined analytically and is given by[12]

$$p_0 = \left[\sum_{k=0}^{n} \frac{n!}{(n-k)!} \rho^k \right]^{-1} \tag{2.4}$$

In Figure 2.2, $\overline{W}$ as a function of n with ρ^{-1} having a value of 40 is plotted. Kleinrock[13] defined the *saturation number*, which we denote by n^*, as

$$n^* = 1 + \frac{1}{\rho} \tag{2.5}$$

Indeed, n^* is the maximum number of sources for which requests can be scheduled without interference.

3. Parallel Best-First Branch-and-Bound Algorithms with a Global OPEN list

In this paper, we assume that the best-first branch-and-bound algorithms are parallelized on a tightly-coupled multiprocessor. A global OPEN list, which contains the nodes that have been generated but not yet examined in the search tree, is shared by all the processors. Each node in the OPEN list is a representation of

the root of a subproblem. Initially the OPEN list contains only a representation of the entire solution space. Each processor then repeatedly removes the best cost node from the OPEN list, solves the node if it is a solution or decomposes the node into one or more child nodes, estimates their cost (and possibly eliminates some of them by applying the dominance relation), and then inserts them into the global OPEN list in the appropriate positions. The basic task of each processor is regular iterations of deletion-decomposition or decomposition-insertion. We will refer to an entire operation of deletion-decomposition or decomposition-insertion as one *iteration*.

The time spent to insert/delete a node into/from the OPEN list depends on the length of the OPEN list and the discipline used. The length of the OPEN list can become very large and hence the time taken to delete from and insert into the OPEN list will not be negligible when compared to the time spent in decomposition. A processor trying to insert/delete a node into/from the OPEN list cannot proceed when any other processor is holding the OPEN list. This introduces a synchronization delay which is ubiquitous in realization of parallel algorithms. Multiprocessor systems usually provide some locking mechanisms, e.g. *spin lock* and *semaphores*[5], for users to implement mutually exclusive codes. Spin lock or busy-waiting is efficient for infrequent and short locking. However, for frequent lockings which take nonnegligible time, spin lock can result in serious lose of memory bandwidth and network bandwidth. Moreover if the accessing order cannot be preserved, programs may execute for arbitrary times since the performance of heuristic search strongly depends on the order of node examination. On the other hand, semaphores are recommended for synchronization of the

OPEN list, in spite of the possible inefficiency due to frequent context switching.

The termination of parallel branch-and-bound algorithms must be handled carefully. To ensure correctness, a *current best solution* (sometimes referred to as the *incumbent* in the literature) is kept in a globally shared memory. The parallel branch-and-bound algorithm is terminated when either one of the following conditions is satisfied: (a) There are no active processes **and** the OPEN list is empty. (b) There are no active processes **and** no nodes in the OPEN list have better cost than the current best solution. In either case the current best solution (if any) is the global solution. Note that the order of the evaluation of the arguments to the Boolean operator **and** above is also important. Here we assume the arguments of **and** are evaluated from left to right. In condition (a), if the evaluation order of **and** is from right to left then it is possible that the OPEN list is tested true before some active process inserts nodes into the OPEN list and terminates. Similar arguments hold for condition (b).

The parallelizing scheme we assume here was first proposed by Imei et al[11]. Similar methods were later used by others[20,24] in implementing best-first branch-and-bound algorithms on shared memory multiprocessors.

4. A Tight Upper Bound for the Speedup of Parallel Best-First Branch-and-Bound Algorithms

In this section, we show how the finite population queueing model can be used to predict the speedup of parallel best-first branch-and-bound algorithms.

Recall that in the above parallel branch-and-bound algorithm, each processor repeatedly decomposes a node and performs OPEN insertion or deletion. The

OPEN list can at most be accessed by one processor; the other conflicting processors must be blocked. It is not difficult to recognize the analogy between the parallel branch-and-bound algorithm and the finite population queueing model discussed in Section 2. The processors are the finite sources of this queueing model. Each processor spends some time (thinking time) decomposing a node and then sends a request (insertion/deletion) to the OPEN list. A FCFS queue can be used to handle the requests at the OPEN list. The time spent in insertion/deletion can be considered as the central server's processing time. Note that there is no processor dedicated to be the central server.

Let us again assume that the average time spent for a processor to decompose a node is λ^{-1} sec. and the time taken to insert/delete a node into/from the OPEN list, when it is free, has an average of μ^{-1} sec. The node decomposition time will not be affected by other processes running concurrently, while the insertion/deletion time could be delayed by other processes trying to access the OPEN list at the same time. Therefore, the average time spent during an iteration (decomposition-insertion/decomposition-deletion) is $\lambda^{-1} + \mu^{-1}$ in the sequential case and $\lambda^{-1} + \overline{W}$ in the parallel case, where $\overline{W}$ is the average response time predicted by Equation (2.1).

Define I_1 as the total number of iterations executed in a best-first branch-and-bound algorithm when a single processor is used and I_n the total number of iterations executed when n processors are used. If detrimental or acceleration anomalies do not exist, I_n is at least as large as I_1 and they are very close when the problem size is large[17,18,23].

The speedup $S(n)$ for n processors is the ratio between the execution times when using one processor and using n processors. Hence,

$$S(n) = \frac{I_1 \times (\lambda^{-1} + \mu^{-1})}{\frac{I_n \times (\lambda^{-1} + \overline{W}(n))}{n}}$$

$$\leq n \times \frac{\lambda^{-1} + \mu^{-1}}{\lambda^{-1} + \overline{W}(n)} \tag{4.1}$$

where $\overline{W}(n)$ is the average response time of the requests at the OPEN list when n processors are used. Using Equation (2.3) as an asymptotic approximation for $\overline{W}$ in Equation (4.1) and letting $\rho = \frac{\lambda}{\mu}$, we obtain

$$S(n) < n \times \frac{1 + \rho}{1 + (n - \frac{1}{\rho})\rho}$$

or

$$S(n) < 1 + \frac{1}{\rho} \tag{4.2}$$

In Figure 4.1, speedup S as a function of n is plotted with a ρ^{-1} value of 40. Note that the speedup saturates very quickly when the number of processors exceeds the saturation number defined in Equation (2.5) though we have a nearly linear speedup before that many processors are used. Here, the saturation number = 41.

Equation (4.1) is obtained with the following assumptions:

a) Equal work between sequential algorithm and parallel algorithm, i.e. $I_1 = I_n$.

b) Decomposable computation is fully parallelized among available processors.

c) No process scheduling overhead.

d) No overhead of locking primitives.

e) No hardware level contention, e.g. memory contention or switch contention.

These assumptions will not be true in practice but can be achieved quite closely when not many processors are used. Hence the result of Equation (4.1) can be considered as a tight upper bound.

5. A Simulation Result

To empirically demonstrate this analysis, a parallel branch-and-bound algorithm was implemented on the BBN Butterfly[2] using Butterfly Lisp[27] to solve the Traveling Salesman Problem. Given a set of cities and the distances between each pair of cities, the TSP is to find a complete shortest tour which visits every city once and only once. Mohan[20] solved the TSP on the Cm* multiprocessor. Although the speedup he obtained was less than 8 when 16 processors were used, he estimated that an almost linear speedup could be achieved for up to 12 processors if a hardware contention problem (referred to as *cluster contention* in Cm*) could be factored out. Rao et al.[24] recently reported an encouraging result in which they obtained a speedup of 7 on 8 processors when solving the TSP on a Sequent Balance 8000. The parallelizing strategy that Mohan and Rao et al. used was basically the same as that illustrated in Section 2.

We did not use Little et al.'s algorithm[19] as both Mohan and Rao et al. did. We used an algorithm based on the *assignment problem*[22]. Let c_{ij} be the distance between city i and city j. Then the m-city TSP can be stated as:

minimize

$$\sum_{j=1}^{j=m} \sum_{i=1}^{i=m} c_{ij} x_{ij}$$

subject to

$$x_{ij} = 0 \text{ or } 1, \quad 1 \leq i \leq m, 1 \leq j \leq m \tag{5.1}$$

$$\sum_{i=1}^{i=m} x_{ij} = 1, \qquad 1 \leq j \leq m \tag{5.2}$$

$$\sum_{j=1}^{j=m} x_{ij} = 1, \qquad 1 \leq i \leq m \tag{5.3}$$

If $x_{ij} = 1$, the route from city i to city j is chosen in the tour. Constraints (5.2) and (5.3) state that there can be only one chosen route which departs from or comes into each city. However, constraints (5.1), (5.2) and (5.3) do not fully characterize the solution of the TSP yet, since the solution so obtained may contain several disjoint cycles. Another constraint must be added to exclude those solutions which contain disjoint cycles.

Figure 5.1 shows an example of a 4-city TSP with cost matrix

$$\begin{bmatrix} \infty & 20 & 30 & 10 \\ 5 & \infty & 7 & 14 \\ 8 & 12 & \infty & 3 \\ 15 & 6 & 11 & \infty \end{bmatrix}$$

The algorithm starts by considering the alternative routes departing from city 1, then city 2, and so forth. The state of a node in the search tree is represented as a tuple $(x_{1i_1}, x_{2i_2}, \ldots, x_{ki_k})$. The cost of the node is the sum of g and h, where $g = c_{1i_1} + c_{2i_2} + \cdots + c_{ki_k}$ and $h =$ the sum of the minimum of each column of the $(m-k)\times(m-k)$ matrix obtained by deleting rows 1 through k and columns i_1 through i_k of the cost matrix.

The OPEN list is organized as a *heap*[9] in our implementation. Figure 5.2

shows the speedup curve we obtained to solve a 12-city TSP. The speedup is almost linear up to 8 processors but only reaches a peak of 12 when 20 processors are used. In general, the number of iterations executed is slightly higher when more processors are used. In Figure 5.3 the speedup curve is adjusted by the number of iterations. That is, if $S(n)$ is the measured speedup for n processors then the adjusted speedup $S(n)^*$ is given by

$$S(n)^* = S(n) \times \frac{I_n}{I_1}$$

The decomposition time and the OPEN insertion/deletion time were sampled during execution when 8 processors were used. Figure 5.4 and Figure 5.5 show the histograms of the OPEN insertion/deletion time and the decomposition time distributions‡. According to the mean decomposition time and mean insertion/deletion time, the ρ^{-1} value is 14.6. The speedup should saturate at 15.6 by our analysis. However, readers should not take the figures we obtained too seriously. Some of the decomposition time measured actually includes the time spent by the incremental garbage collector[4] in the Blisp system† , and hence the mean decomposition time measured will be slightly higher than it should be. In addition, the semaphores we used§ contain an internal critical region which also introduces non-negligible synchronization delay.

Note that higher speedup can be expected when solving a TSP with larger number of cities. This is because the decomposition time is proportional to the

‡ The BBN Blisp we used was a beta release from BBN. No compiler was available when this paper was prepared. The interpreter was very slow in absolute terms.

† This can be seen from the extremely high time occurrences in Figure 5.5.

§ The semaphores we used was implemented using *futures* in Blisp. Futures are very expensive in the current Blisp version.

problem size and larger mean decomposition time will increase the value of ρ^{-1}.

6. Discussion and Conclusion

We have demonstrated that the finite population queueing model can be used to accurately analyze the speedup of parallel branch-and-bound algorithms. The accuracy is obtained because the synchronization delay is taken into consideration. Certainly, more accurate analysis should include architectural inefficiency, e.g. memory contention, bus contention, switch contention, etc., and system overhead, e.g. scheduling overhead, overhead of synchronization primitives, etc. However, this kind of analysis is machine-dependent and its long-term significance on the design of parallel algorithms is unclear. There is hope that new technology will reduce the architectural inefficiency and system overhead to an insignificant amount[25].

The results we obtained from theoretical analysis and simulation show that because of the synchronization delay, the speedup of parallel algorithms can saturate very quickly after a nearly linear speedup up to a certain number of processors. This apparently pessimistic result can be used to design more efficient parallel algorithms. The saturation number defined in Equation (2.5) can be considered as the maximum number of processors to be used before speedup saturates. Beyond that parallelism does not help. Extending the saturation point is equivalent to enlarging the value of ρ^{-1}. This can be achieved by increasing λ^{-1} or decreasing μ^{-1}. We could raise λ^{-1} by magnifying the *granularity* of the node decomposition. One possible way is to execute more iterations before a process communicates with the global OPEN list. For heuristic search, this must

be done carefully since larger granularity means less heuristic-guidance. μ^{-1} can be lowered by better insertion and deletion disciplines for the global OPEN list. For example, heap insertion and deletion is superior to linear insertion and deletion.

Other optimizing methods are also possible. The OPEN list can be partitioned into several disjoint regions according to different ranges of cost estimation values. An extreme case is that each region corresponds to a single cost estimation value so that insertion and deletion can be done in O(1) time. This method was actually used by Rao et al.[24] to solve a 15-puzzle problem. They took advantage of the fact that the cost estimation function has a small range and can be predetermined. Since now each region is locked separately the OPEN list bottleneck could be potentially relieved by a degree of a multiple. The performance of the algorithms based on a multiple-region OPEN list can be formally analyzed by a more general *finite source multiple server queueing model.*

If the range of the the cost estimation function is not small or cannot be predetermined, a possible way to alleviate the bottleneck is to organize the global OPEN list as a *concurrent* B-tree[1,16] so that insertions and deletions performed in different subtrees can be possibly executed in parallel.

Finally we note that a similar analysis can be applied to parallel AI systems whose main communication medium is through a global shared database, e.g. a *blackboard*[6].

ACKNOWLEDGMENT

The authors are deeply grateful to Dr. Udaya Shankar and Dr. Satish Tripathi for their critical review of the first draft of this paper and their valuable comments on it.

REFERENCES

1. Bayer, R. and Schkolnick, M., *Concurrency of operations on B-trees,* Acta Informatica, **9**, 1977, pp. 1–21.

2. BBN Laboratories, *Butterfly Parallel Processor Overview,* Bolt, Beranek and Newman, Cambridge Massachusetts, Dec. 1985.

3. Coffman, E. G. and Denning, P. J., *Operating Systems Theory,* Englewood Cliffs, N.J.: Prentice-Hall, 1973.

4. Courtemanche, A., *MultiTrash, a parallel garbage collector for MultiScheme,* M.I.T. B.S.E.E. Thesis, Cambridge, Massachusettes, January 1986.

5. Dijkstra, D. W., *The structure of THE multiprogramming system,* Comm. ACM, **11**, 1968, pp. 341–346.

6. Erman, L. and Lesser, V., *A Multi-Level Organization for Problem-Solving Using Many Diverse Cooperating Sources of Knowledge,* International Joint Conference on Artificial Intelligence, 1975, pp. 483–490.

7. Fennel, R. D. and Lesser, V. R., *Parallelism in AI problem solving: A case*

study of Hearsay II, IEEE Trans. on Computers, Vol. **C-26**, 1977, pp. 98–111.

8. Fuller, S. H., *Performance evaluation,* Chap. 11 in *Introduction to Computer Architecture,* H. S. Stone, ed., SRA, 1980, pp. 527–586.

9. Horowitz, E. and Sahni, S., *Fundamentals of Computer Algorithms,* Rockville, MD: Computer Science Press, 1978.

10. Ibaraki, T., *Theoretical comparisons of search strategies in branch-and-bound algorithms,* Int. J. of Comp. and Inf. Sci., **5**, 1976, pp. 315–344.

11. Imai, M. and Yoshida, Y., *A parallel search scheme for multiprocessor systems and its application to combinatorial problems,* Sixth International Joint Conference on Artificial Intelligence, August 1979, pp. 416–418.

12. Kleinrock, L. *Queueing systems: theory,* Vol I, New York: Wiley, 1975.

13. Kleinrock, L. *Queueing systems: computer applications,* Vol II, New York: Wiley, 1975.

14. Lai, T. H. and Sahni, S., *Anomalies in parallel branch-and-bound algorithms,* Comm. ACM, **27**, 1984, pp. 594–602.

15. Lawer E. L. and Wood D. E., *Branch-and-bound methods: a survey,* Operations Research, **14**, 1966, pp. 699–719.

16. Lehman, P. L. and Yao, S. B., *Efficient locking for concurrent operations on B-trees,* ACM Trans. on Database Systems, **6**, 1981, pp. 650–670.

17. Li, G. -J. and Wah, B. W., *Computational efficiency of parallel approximation branch-and-bound algorithms,* International Conference on Parallel Pro-

cessing, 1984, pp.473–480.

18. Li, G. -J. and Wah, B. W., *Coping with anomalies in parallel branch-and-bound algorithms* IEEE Trans. Computers, **C-35**, 1986, pp. 568–573.

19. Little, J., Murty, K., Sweeny, D. and Karel, C., *An algorithm for the traveling salesman problem*, Operations Research, **11**, 1963, pp. 972–989.

20. Mohan, J., *A study in parallel computation—the Traveling Salesman Problem*, Technical Report CMU-CS-82-136, Computer Science Department, Carnegie-Mellon University, August, 1982.

21. Nillsson, N. J., *Principles of Artificial Intelligence*, Palo Alto, CA: Tioga Press, 1980.

22. Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Englewood Cliffs, N.J.: Prentice-Ha.., 19 . 439–441.

23. Quinn, M. J., and Deo, N., *An upper bound for the speedup of parallel best-first branch-and-bound algorithms*, BIT **26**, 1986, pp. 35–43.

24. Rao, V. N., Kumar, V. and Ramesh K., *Parallel heuristic search on a shared memory multiprocessor*, Technical Report AI TR87-45, Artificial Intelligence Laboratory, University of Texas, Austin, Jan. 1987.

25. Rettberg, R. and Thomas, R. *Contention is no obstacle to shared-memory multiprocessing*, Comm. ACM, **29**, 1986, pp. 1202–1212.

26. Sherr, A. A., *An Analysis of Time-Shared Computer Systems*, Cambridge, MA: MIT Press, 1967.

27. Steinberg, S. A., Allen, D., Bagnall, L. and Scott, C., *The Butterfly Lisp System,* AAAI, 1986, pp. 730–734.

28. Mitten, L. *Branch-and-bound methods: general formulation and properties,* Operations Research, **14**, 1970, pp. 24–34.

29. Wah, B. W. and Eva Ma, Y. W., *MANIP—A multicomputer architecture for solving combinatorial extremum-search problems,* IEEE Trans. on Computers, **C-33**, May 1984, pp. 377–390.

30. Tripathi, S. K. *On detecting parallelism in software,* J. Systems and Software, **6**, 1986, pp. 133–135.

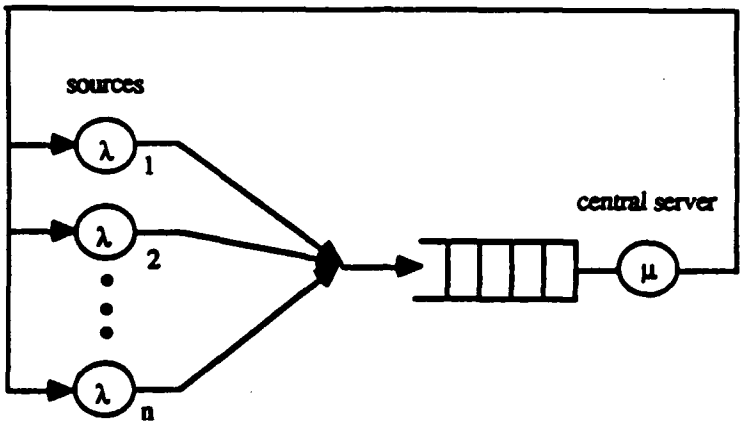


Figure 2.1. Finite source queuing structure

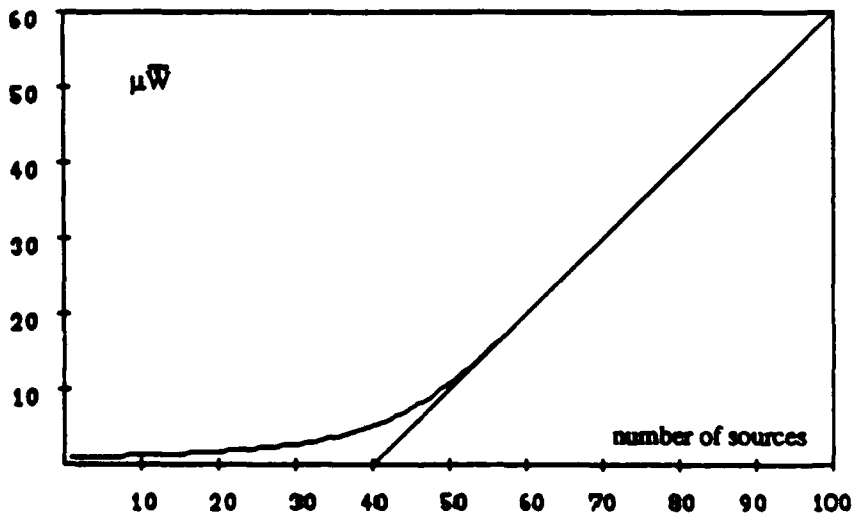


Figure 2.2 Average response time curve obtained from Equations (2.1) and (2.3) with $1/\rho = 40$.

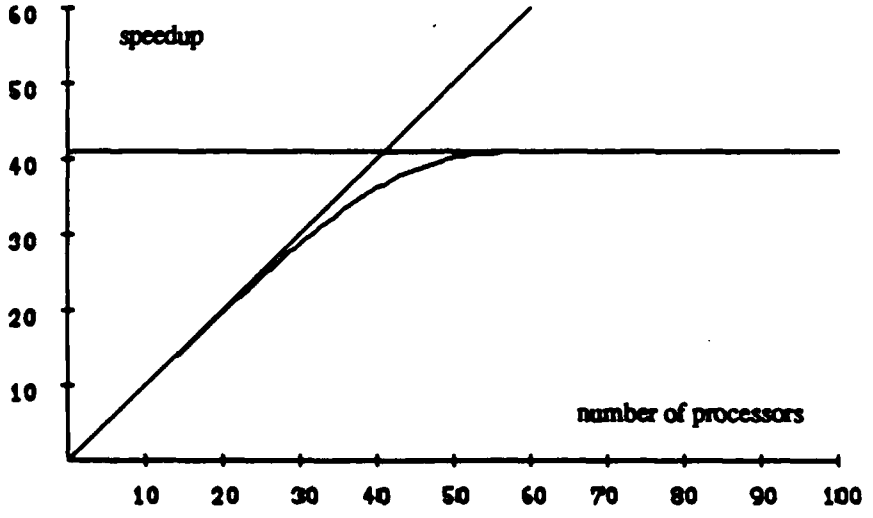


Figure 4.1 Speedup curve obtained from Equation (4.1) with $1/\rho = 40$.

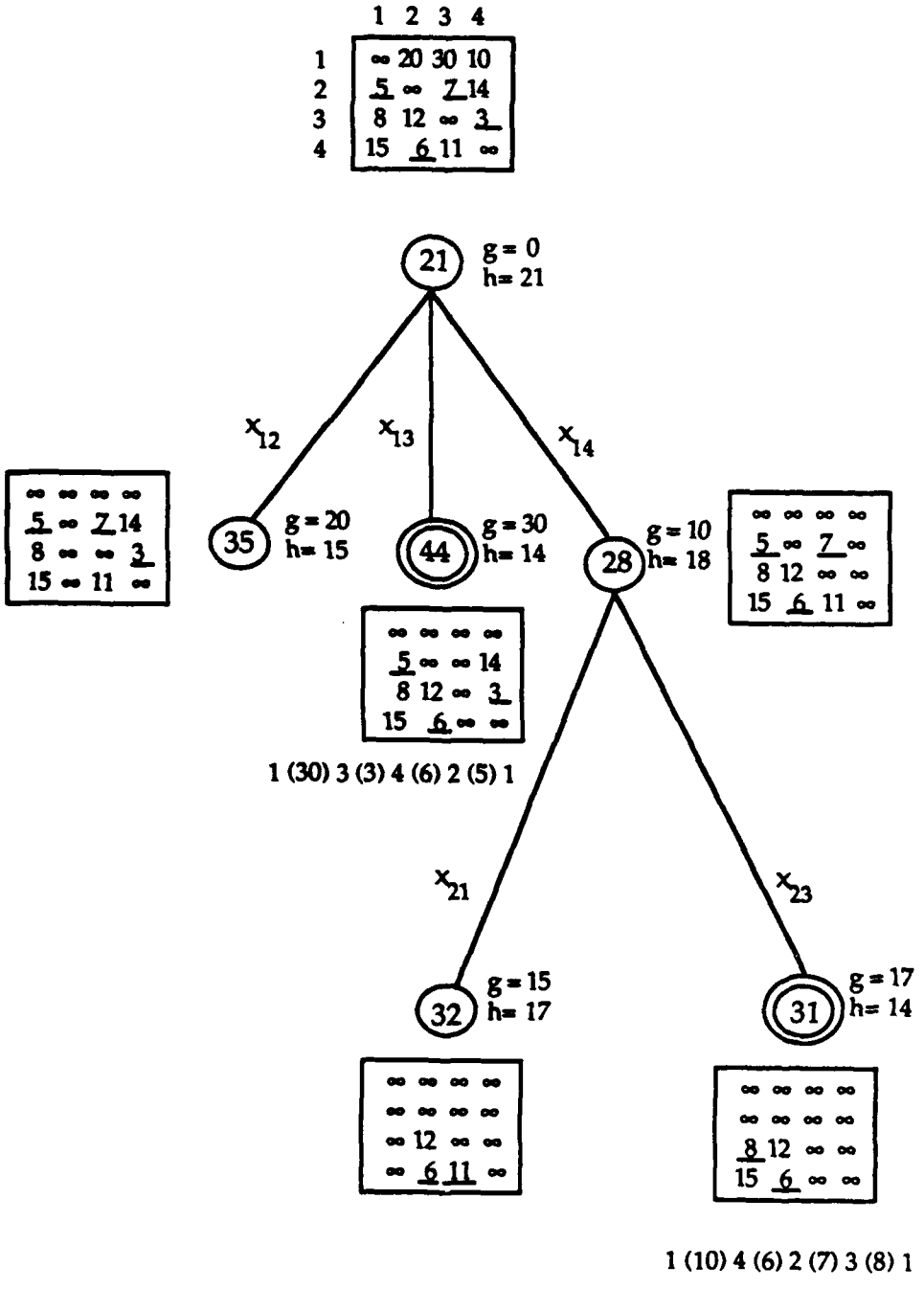


Figure 5.1 An example of a 4-city TSP.

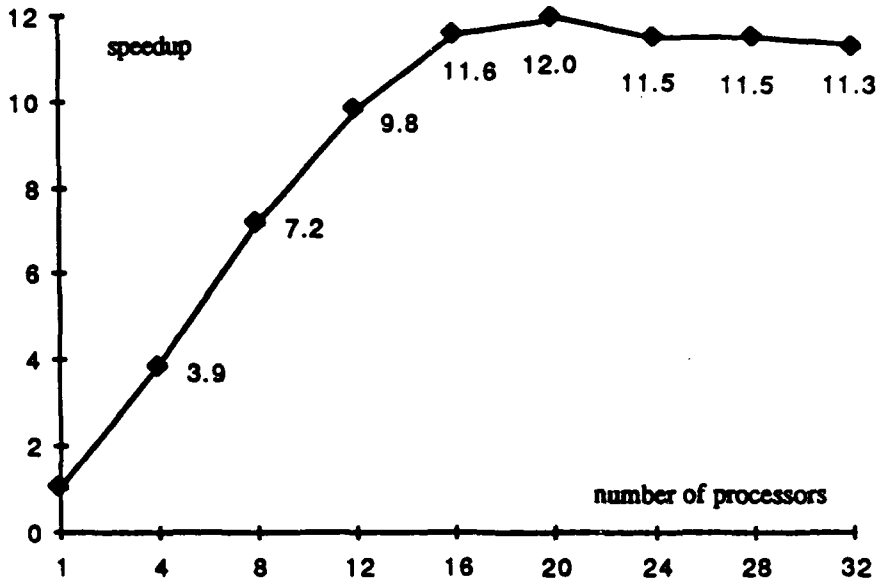


Figure 5.2 Actual speedup curve for a TSP with 12 cities

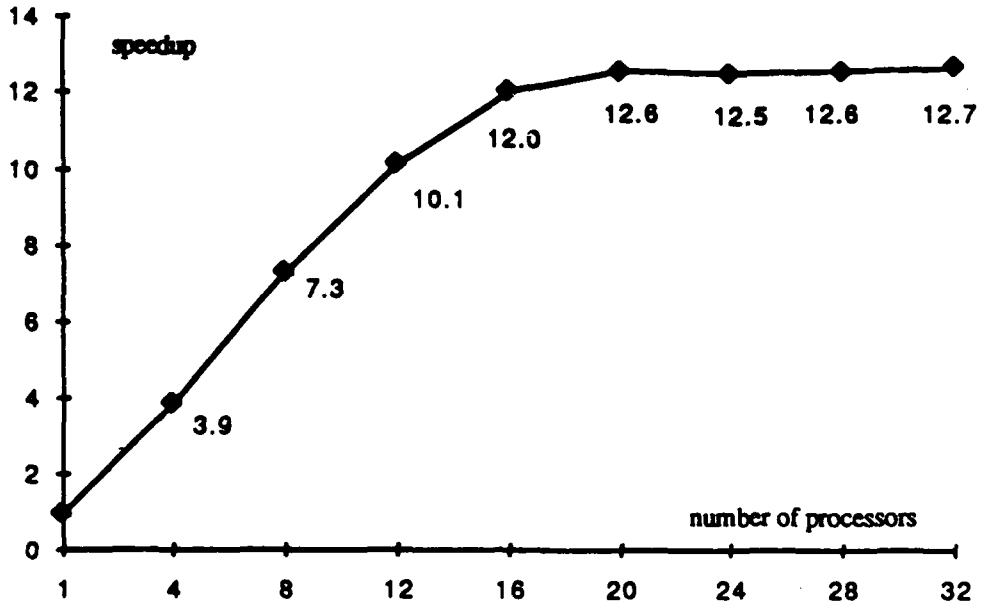


Figure 5.3 Adjusted speedup curve from Figure 5.2 by the number of iterations.

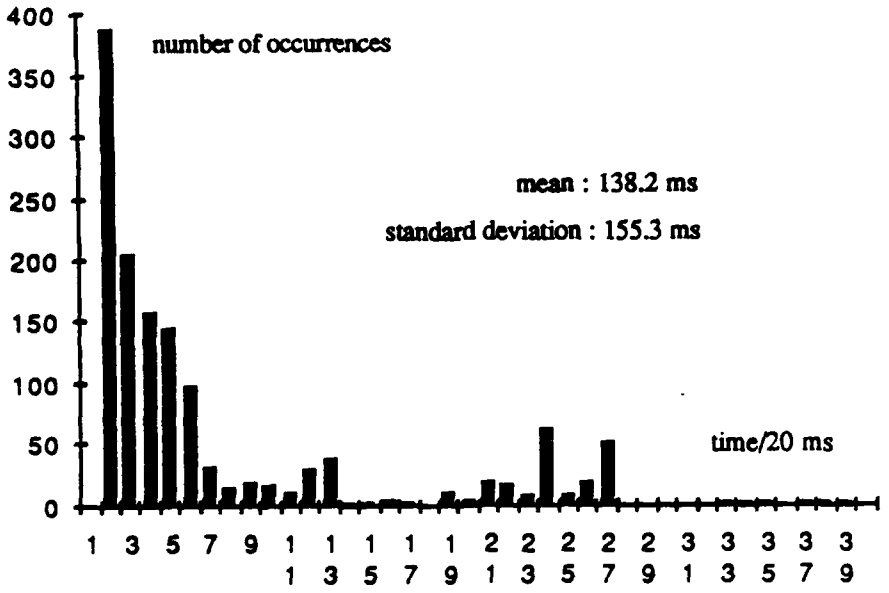


Figure 5.4 Histogram of insertion/deletion time.

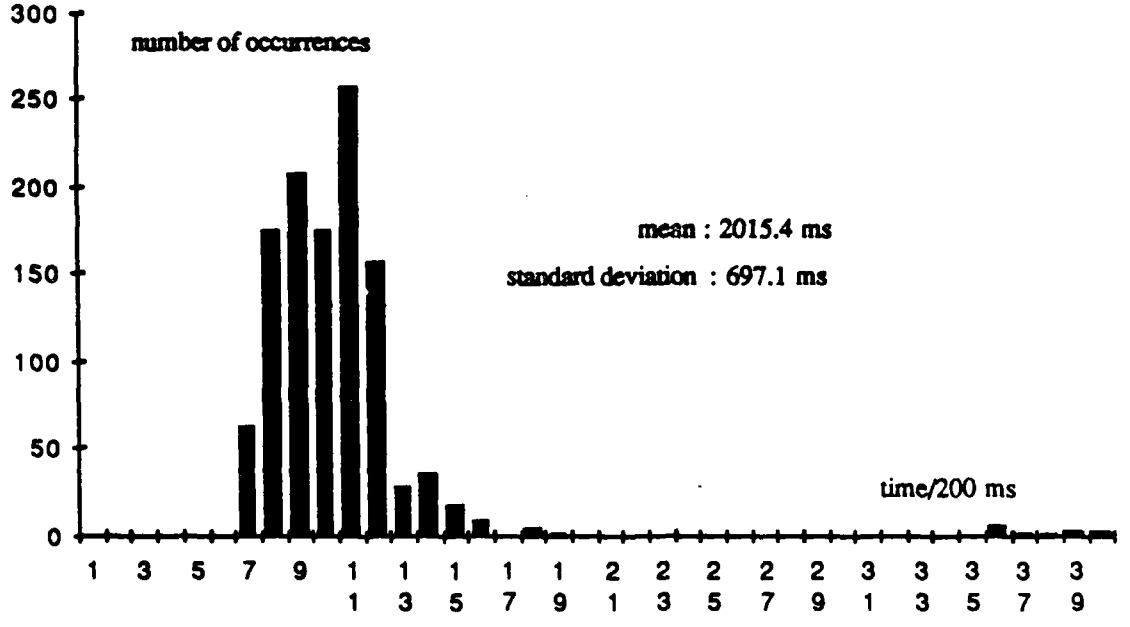


Figure 5.5 Histogram of decomposition time.

END

9-87

DTIC